UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/693,630	10/23/2003	Sriram Subramanian	4120	1530

7590 06/05/2006	EXAMINER
Law Offices of Albert S. Michalik, PLLC	WOODS, ERIC V

	ART UNIT	PAPER NUMBER
Law Offices of Albert S. Michalik, PLLC	2628	

Law Offices of Albert S. Michalik, PLLC
Suite 193
704 -228th Avenue NE
Sammamish, WA 98074

DATE MAILED: 06/05/2006

Please find below and/or attached an Office communication concerning this application or proceeding.

	Application No.	Applicant(s)
Office Action Summary	10/693,630	SUBRAMANIAN ET AL
	Examiner	Art Unit	
	Eric Woods	2628	

-- *The MAILING DATE of this communication appears on the cover sheet with the correspondence address* --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE _3_ MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on _26 January 2006_.

2a)☐ This action is **FINAL.**   2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) _1-67_ is/are pending in the application.

    4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) _1-67_ is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

    Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

    Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

    a)☐ All   b)☐ Some * c)☐ None of:

      1.☐ Certified copies of the priority documents have been received.

      2.☐ Certified copies of the priority documents have been received in Application No. _____.

      3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

    * See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1)☒ Notice of References Cited (PTO-892)

2)☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)

3)☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08)
    Paper No(s)/Mail Date _____.

4)☐ Interview Summary (PTO-413)
    Paper No(s)/Mail Date. _____.

5)☐ Notice of Informal Patent Application (PTO-152)

6)☐ Other: _____.

DETAILED ACTION

Response to Amendment

The affidavits under 37 CFR 1.132 filed 20 January 2006 is sufficient to overcome the rejection of claims 1-67 based upon the David reference. The affidavits serve to show that such reference is invention 'by another' as required by the statute under 35 USC 102(e).

The terminal disclaimer further serves as an affidavit under 37 CFR 1.130 that the work is commonly assigned and eligible for the exclusion under 35 USC 103(c)(1).

Terminal Disclaimer

The terminal disclaimer filed on 20 January 2006 disclaiming the terminal portion of any patent granted on this application which would extend beyond the expiration date of co-pending application 10/693,673 has been reviewed and is accepted. The terminal disclaimer has been recorded.

Double Patenting

Claim 1 is provisionally rejected on the ground of nonstatutory obviousness-type double patenting as being unpatentable over claim 1 of copending Application No. 10/693633. Although the conflicting claims are not identical, they are not patentably distinct from each other because the claim scope is concomitant, and each has a small bit more specificity than the other.

This is a provisional obviousness-type double patenting rejection because the conflicting claims have not in fact been patented.

Claim 1, instant application:	Claim 1, 10/693,633 (as originally filed)
A computer-implemented method for arranging computer graphics data for processing into an output, comprising:	In a computing environment, a method comprising:
-Receiving a function call via an application programming interface of a media integration layer, the function call corresponding to graphics-related data; and	-Receiving a function call via an interface, the function call comprising markup language data; and
-Causing data in a scene graph data structure to be modified based on the function call.	-Interpreting the markup language data to cause data in a scene graph to be modified.

Claim 1, instant application:	Claim 1, 10/693,633 (as of 9/27/05)
A computer-implemented method for arranging computer graphics data for processing into an output, comprising:	In a computing environment, a computer-implemented method comprising,
-Receiving a function call via an application programming interface of a media integration layer, the function call corresponding to graphics-related data;	-Receiving a function call via an application programming interface, the function call comprising markup language data;

and -Causing data in a scene graph data structure to be modified based on the function call.	-Interpreting the markup language data to cause data in a scene graph to be modified; and -Causing a change in a display in response to the modification of data in the scene graph.

In the '633 application, the markup language data is all expressly related to graphics, specifically generating and changing representations within the scene graph. Further, "interpreting markup language" to cause data to be modified still causes data to be modified. Therefore, the claims are essentially duplicates.

Claims 1-67 of this application conflict with claims 1-67 of Application No. 10/693,633. 37 CFR 1.78(b) provides that when two or more applications filed by the same applicant contain conflicting claims, elimination of such claims from all but one application may be required in the absence of good and sufficient reason for their retention during pendency in more than one application. Applicant is required to either cancel the conflicting claims from all but one application or maintain a clear line of demarcation between the applications. See MPEP § 822.

Examiner is now issuing a requirement that all redundant claim(s) be removed from the instant application. The 10/693,633 application is farther along in prosecution than the current application and thusly receives priority in this matter.

Response to Arguments

Applicant's arguments, see Remarks and various other papers, filed 20 January

2006, with respect to the rejection(s) of claim(s) 1-67 under 35 USC 102(e) and various

other statutes have been fully considered and are persuasive.

Therefore, the provisional obviousness-type double patenting rejection has been

withdrawn in view of the terminal disclaimer.

The rejections under 35 USC 102(e) stand withdrawn since applicant has filed

affidavits under 37 CFR 1.132 clearly showing that the invention is not 'by another' and

thusly excluding that art from further consideration.

However, upon further consideration, a new ground(s) of rejection is made in

view of various references as below.

Definitions

The term "media integration layer" is defined in the instant specification (PGPub)

in [0059-0060] to be an architecture that includes immediate mode graphics application

programming interfaces, screen-partitioning data structures, and an API. The MIL is a

composited system that works in layers. (Quoting from those 2 paragraphs). In short,

applicant defines a MIL to merely be the graphics software operating a computer and

receiving function calls.

Therefore, as such, examiner stipulates that the term is expansively broad and

covers all graphics architectures, etc. Any graphics subsystem serves as a 'media

integration layer'. The specification is **not** enabling for a more narrow reading of the

term. Please note that this interpretation is a matter of sufficiency of disclosure and

written description. Please note that it cannot be reviewed under 37 CFR 1.144; rather,

35 USC 134(a) is the relevant statute. Further, it cannot be subject to review by itself but must be reviewed in the context of the other rejections below, since this stipulation is **not** a ground of rejection *per se*. Please further note that a failure to respond to this point will be regarded as a full waiver upon appeal, precluding any future use and barring future discussion in any subsequent agency action(s) or legal proceeding(s) concerning the matter in question.

It is noted, however, that the specification does require that the 'media integration layer' utilize markup language, and that the functional calls are or can be embodied as markup language. [Please see PGPub 0059-0061].

It should also be noted that although the instant application never expressly states that it is attempting to patent the SVG (Scalable Vector Graphics) specification that it is very much trying to do so, as most of the dependent claims match up to elements within the SVG standard. Next, it should be noted that in the PGPub in paragraph [0009] states that the interfaces support "a vector graphics markup language" e.g. SVG. Therefore, examiner is supported by a preponderance of the evidence in concluding that this application and its related, co-pending applications such as 10/693,633, and the one that was used as a reference in the last action and terminally disclaimed are attempting to patent the SVG standard, browsers, parsers, file formats, etc.

Duty of Disclosure

Applicant is reminded that under 37 CFR 1.56, applicant is required to submit all material known to be relevant to the patentability of the subject matter within this

application. Applicant has not disclosed anywhere in the instant application the existence of the SVG standard or any details concerning it. Applicant is requested to therefore submit all known documentation that is associated with the development of the present invention in terms of known prior art on an IDS under 37 CFR 1.97 and 1.98. Note 37 CFR 1.56(a)(2). **Failure to do so will result in the issuance of a requirement to compel the submission of such information via 37 CFR 1.105.**

Applicant is further reminded that based on the co-pending applications that the instant Examiner is examining that share applicants and inventive entities, applicant has not contested the point that the instant application is (at the very least) very similar to the SVG standard / specification. Finally, no IDS documents have been submitted in the instant application, where there have been several relevant references cited in the prosecution of the related, co-pending applications, which share common assignees, inventive entities, and applicant's representatives, that should be submitted in an IDS, where the references cited during the prosecution of such related applications are and would be clearly material as per the standards set forth in 37 CFR 1.56(b)(1) and 1.56(b)(2)(i)-(ii), particularly with respect to 37 CFR 1.56(b)(2)(ii).

The burden is on applicant, not the Office, to ensure proper submission of all related documents. Once the examiner has pointed out the existence of related applications that examiner deems relevant and requests the submission of all such references, there can be no question whatsoever that the Office has satisfied its portion of any burden to obtain such information.

Claim Objections

Claim 2 is objected to under 37 CFR 1.75 as being a substantial duplicate of

claim 4. When two claims in an application are duplicates or else are so close in

content that they both cover the same thing, despite a slight difference in wording, it is

proper to object to the other as being a substantial duplicate of the allowed claim. See

MPEP § 706.03(k).

Claim Rejections - 35 USC § 101

35 U.S.C. 101 reads as follows:

Whoever invents or discovers any new and useful process, machine, manufacture, or composition of
matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the
conditions and requirements of this title.

Claims 1-67 are rejected under 35 U.S.C. 101 because they are directed to non-

statutory subject matter for two reasons. First, the method and system claims are both

embodied by computer programs, which are not claimed (see instant specification, page

13, line 15, through page 12, line 14). Such computer programs are further embodied

via carrier wave (that is, the computer-readable media upon which such programs are

embodied specifically includes carrier waves, the Internet, etc), which are prima facie

non-statutory as per the Interim Guidelines for Patent Subject Matter Eligibility.

Next, the claims do not produce a 'concrete, tangible, and practical' result as

required by *State Street.* The claims do not take in data, but rather merely manipulate

data within a computer. A functional call is merely electronic instructions within a

computer, and the data in the scene graph that may be modified is a data structure per

se, which is also within the memory of a computer. As such, the claims merely embody

the quintessential abstract idea and do not produce any practical results. Applicant is

directed to *In re Alappat*, where the method of modifying graphic output was found to be statutory precisely because it **modified graphical output**, that is, it produced concrete and tangible results that had a practical effect (reducing anti-aliasing). Next, the claims do not produce any results, relied upon by regulators or not (*State Street, AT&T v. Excel Comm.*, etc).

There is no result whatsoever from the operation of such claims. Causing a computer to modify data that is solely within its memory that has no connection to real-world applications (such as pre- or post-computer activity linked to a real world application) only constitutes the quintessential abstract idea and thusly not statutory. The claims need to recite some kind of practical application, such as outputting the modified scene graph data to a monitor or displaying it in some manner.

The claims rejected above under 35 USC 101 as nonstatutory are further rejected under various other statutes as set forth below in anticipation of applicant amending the claims to place them within one of the four statutory classes of subject matter.

Claim Rejections - 35 USC § 112

The following is a quotation of the first paragraph of 35 U.S.C. 112:

> The specification shall contain a written description of the invention, and of the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the art to which it pertains, or with which it is most nearly connected, to make and use the same and shall set forth the best mode contemplated by the inventor of carrying out his invention.

Claims 1-36 are rejected under 35 U.S.C. 112, first paragraph, as based on a disclosure which is not enabling. A computer, which is critical or essential to the practice of the invention, but not included in the claim(s) is not enabled by the

disclosure. See *In re Mayhew*, 527 F.2d 1229, 188 USPQ 356 (CCPA 1976).

Specifically, a computer is obviously necessary to further process computer graphics

data, but the claim does not recite this. The preamble of the claim needs to be

amended to read: 'A **Computer-implemented** method...'

The following is a quotation of the second paragraph of 35 U.S.C. 112:

The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

Claims 3, 5, 11, 15, 17-19, 23, 26-27, 33, 37, 39-40, 43, 59, and 65 are rejected

under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly

point out and distinctly claim the subject matter which applicant regards as the

invention. See below for specific listing of claims

Specifically, claims 3 and 17-19 are rejected because the term "a visual" is used

where it is unclear what the terminology is referring to and the specification does not

reasonably apprise one of ordinary skill in the art what the definition would be. **For

examination purposes, examiner will treat "a visual" as meaning "a visual object or

element" in the context of a visual element in SVG specification, which applicant clearly

states is being used in his invention.

Claims 15 and 43 are rejected because the term 'path' is used, and it is unclear

what the meaning of 'path' – whether applicant is referring to the path of a visual

element during an animation or a path as defined in the SVG specification.

Claims 3, 5, 11, 20, 33, and 65 are rejected because the term 'interface' is used,

and it is unclear what the intended meaning of 'interface' is in the context of the

specification, because applicant refers to the SVG specification, wherein an 'interface' is

a programming construct – that is, an interface implemented for elements, wherein it is

unclear whether that is intended meaning or a generalized interface (hardware or

software) is intended. Further, the independent claims disclose an API, where the term

interface necessitates another interface, which lacks antecedent basis.

Claims 5, 26, 27, and 59 are rejected because the term 'context' is used, and it is

unclear what the meaning of 'context' is in this claim. In the SVG specification, the term

'context' is not specifically set forth and applicant does not provide a clear basis of the

definition. Context can relate to the device-specific drawing information and

capabilities, as set forth by various references; it can be a term relating specifically to

the properties and metadata associated with various visual elements; it can be many

things. As such, particularly with respect to the graphics art, it is critically important to

be able to determine the implication of the word, because the 'context' of a device for

rendering purposes is very different that the 'context' of a data node in a scene graph,

etc.

Claims 37 and 40 are rejected because they should be dependent on claim 36 as

per the specification, not claim 26. They are also out of order if they are in fact

dependent on claim 26. They are being treated as dependent upon claim 36 for

purposes of examination.

Claim 39 is rejected because the term 'collection object' is used, and it is

unknown how such an object would differ from the 'container object' recited in the

parent claim 36 to this particular claim and the specification does not offer a separate

definition for such an object. The definition adopted by the examiner is listed in the

rejected to the claim.

Claim Rejections - 35 USC § 103

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all

obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set
> forth in section 102 of this title, if the differences between the subject matter sought to be patented and
> the prior art are such that the subject matter as a whole would have been obvious at the time the
> invention was made to a person having ordinary skill in the art to which said subject matter pertains.
> Patentability shall not be negatived by the manner in which the invention was made.

The factual inquiries set forth in *Graham* v. *John Deere Co.*, 383 U.S. 1, 148

USPQ 459 (1966), that are applied for establishing a background for determining

obviousness under 35 U.S.C. 103(a) are summarized as follows:

1. Determining the scope and contents of the prior art.
2. Ascertaining the differences between the prior art and the claims at issue.
3. Resolving the level of ordinary skill in the pertinent art.
4. Considering objective evidence present in the application indicating
   obviousness or nonobviousness.

Claims 1-4, 18, 20, 28, 36, and 62 are rejected under 35 U.S.C. 103(a) as being

unpatentable over Itoh et al (US 6,741,242 B1) in view of the SVG specification (Itoh

references the SVG specification in the list of cited documents on the patent anyway,

and Itoh clearly utilizes SVG as set forth below; therefore, no other statement for why

SVG is incorporated is necessary).

As to claim 1,

A computer-implemented method for arranging computer graphics data for processing

into an output, comprising: (Preamble is not given patentable weight, since it only

recites a summary of the claim and/or an intended use, and the process steps and/or

apparatus components are capable of standing on their own; see Rowe v. Dror, 112

F.3d 473, 42 USPQ2d 1550 (Fed. Cir. 1997), Pitney Bowes, Inc. v. Hewlett-Packard

Co., 182 F.3d 1298, 1305, 51 USPQ2d 1161, 1165 (Fed. Cir. 1999), and the like.)

-Receiving a function call via an application programming interface of a media

integration layer, the function call corresponding to graphics-related data; and (Kim Fig.

2 clearly shows that the system receives 3D data in X3D format, which is known to be in

a markup language [0004-0008]. X3D is known to one of ordinary skill in the art to be

the next generation of the VRML (virtual reality markup language) and to accept and be

an extension of XML (extensible markup language). Next, the X3D browser – element

140 in Figs. 1 and 2, makes function call based on information that it obtains when it

sends out requests for data. Further, see Fig. 3, where clearly the system is shown to

receive user events and data from the user interface. Clearly, these represent function

calls via an interface.)(Itoh teaches in 1:20-2:25 that clearly the system receives markup

language of form similar to XML, and that the system works with a web browser, in a

similar way as that of Kim. Further, Itoh teaches the use of an API (19:1-20, which

teaches the use of a Java AWT API and further teaches that it is well known in the art to

make calls to graphics libraries such as OpenGL (see 19:1-2); 19:50-23, where graphics

array data is passed to the Java AWT API; 22:33-55, where functions are shown to

make calls to the API, e.g. functions are used to set or change parameters, and those

requests are sent to the API, which therefore proves that an API is used to receive a

function call; finally 30:26-52, where it very clearly states that the scene graph provides

an API that can make many changes to the content, which matches with the above

system. **See 22:50-55 – "Here, as described in detail later, a scene graph is data**

for a data structure of a non-cyclic graph composed of a tree-shaped structure of

nodes, and defined as an API in the above function. Therefore, the scene graph of

necessity must possess an API, as must the objects within it. Also, note that Itoh 1:20-

2:50, 22:19-26, etc., it is noted that the three-dimensional source data can be VRML,

and that data can also be obtained in SVG format (2:16-25). Note next 30:33-55, where

it is specified that the scene graph provides an API capable of stipulating group nodes

and leaf nodes, and adding scene attributes which involve context, color of a shape or

texture, and the like. Therefore, clearly the scene graph receives function calls via an

API relating to graphics data, since the API can also affect texture and color changes in

nodes in the scene graph)

-Causing data in a scene graph data structure to be modified based on the function call.

(Itoh teaches in 1:20-2:25 that clearly the system uses a scene graph. Further, Itoh

teaches the use of scene graphs and the modification of their data using markup

language (for example, the listed use VRML in 1:20-2:50, 22:19-26, etc, where XML and

other languages are also taught) (19:1-20, 19:50-23, 22:33-55, finally 30:26-52, where it

very clearly states that the scene graph provides an API that can make many changes

to the content, which matches with the above system), and Itoh can receive data in SVG

format as well (2:19-28). Next as set forth above, the scene graph allows changes to be

made as far as adding textures and the like, which clearly would constitute modifying

data in the scene graph –12:20-34, where such information is associated with a node in

the scene graph, see Figures 12 and 21)

Reference Itoh thusly teaches all the limitations of the recited claim, except that

in light of applicant's specification, it is unclear the exact extent of markup language

required by the term 'media integration layer', and it is unclear if parsing is required,

where if the term is taken broadly Itoh would teach all the limitations, which it is believed

to do.

Reference Itoh does not expressly state that the data in the scene graph is

modified by the function call to the scene graph API. However, it would have been

obvious to one of ordinary skill in the art at the time the invention was made to modify

Itoh to change the data in the scene graph so as to utilize a singular data structure.

As to claims 2 and 4,

The method of claim 1 wherein causing data in the scene graph to be modified

comprises causing initialization of a new instance of a visual class.

The Itoh reference, which teaches that SVG, VRML, etc. data is decomposed

into scene graphs, see Fig. 8, element 101, and 4:10-40, which clearly teaches the use

of scene graphs, and again – whenever new visual elements enter the scene, new

subgroups are instantiated, which *prima facie* (see SVG specification, section 9) are

elements that compose visual objects, which therefore are new instances of a visual

class as recited above, since a class as recited by applicant is comparable to the basic

'shapes' in SVG (applicant's specification clearly uses it – 23:1-20 where applicant's

invention adopts all classes and shapes from SVG) and thusly meets the recited

limitation. Further, in Itoh 29:60-30:25, the leaf nodes are taught to be new instances of

various visible classes and other elements, which would require them to be new

instances of 'visual classes', since the Itoh implementation uses a Java API, and it is well known in the art in Java that all objects are instances of classes, and any object that would appear on the drawing canvas would prima facie be a 'visual object' and thus a new instance of a 'visual class'. Motivation and combination is taken from claim 1.

As to claim 3,

The method of claim 2 further comprising, receiving a functional call via an interface corresponding to a transform associated with the visual.

Itoh teaches this limitation in for example 29:60-30:25, where it is taught that the group node applies transform nodes to map the spatial positions of lower nodes and otherwise to control the elements in the scene graph, and further that (3:40-55) two-dimensional objects can be transformed into three-dimensions. Such modifications prima facie must associate code with a suitable / desired transform (e.g. scaling, rotation, et cetera 11:55-57, where types of transforms are listed), as that is the only way either a hierarchy of nodes (e.g. Fig. 21) or single nodes could be scaled, transformed, etc., and obviously all modifications are done via markup language as recited in the claims.

As to claim 18,

The method of claim 1 wherein causing data in a scene graph data structure to be modified comprises invoking a function related to transforming coordinates of a visual in the scene graph data structure.

Clearly, Itoh as stated above in the rejection to claim 3 teaches in 29:60-30:25 that transform nodes are applied to transforming spatial coordinates of objects in the

scene graph in the form of child nodes and the rest of it. Since only the primary

reference is utilized, no separate combination or motivation is required.

As to claim 20,

The method of claim 1 wherein causing data in the scene graph be modified comprises

invoking a function via a common interface to a visual object in the scene graph data

structure.

Clearly, such modifications to the scene graph are made via the Java API and

the scene graph API itself to allow the objects to be modified as set forth in the rejection

to claim 1 above. Such an API is clearly common and is an interface. As such, the

rejection to claim 1 is incorporated by reference.

As to claim 28,

The method of claim 1 wherein causing data in the scene graph to be modified

comprises invoking a function to place a three-dimensional visual into the scene graph

data structure.

Further Itoh obviously (1:30-2:45) puts three-dimensional objects in the scene

graph (see Fig. 8, 12, 21, etc; 5:45-6:16) using a VRML file, where clearly this would

constitute invoking a function to place a three-dimensional visual into the scene graph,

where this clearly fulfills this limitations of the claim.

As to claim 29,

The method of claim 28 wherein causing data in a scene graph data structure to be

modified comprises mapping a two-dimensional surface onto the three-dimensional

visual.

Clearly, the Itoh reference teaches mapping textures onto three-dimensional objects, which can be nodes in the scene graph, note 2:25-45 ('material of a 3D graphic form, texture, ...) where clearly a material applied to a 3D graphic form is an image applied to a 3D shape, and a texture can also be so applied as noted above (30:1-40 and the like).

As to claims 36 and 65, the rejection to claim 1 is herein incorporated by reference in its entirety.

As to claim 36,

In a computing environment, a computer system comprising: (Preamble is not given patentable weight, since it only recites a summary of the claim and/or an intended use, and the process steps and/or apparatus components are capable of standing on their own; see Rowe v. Dror, 112 F.3d 473, 42 USPQ2d 1550 (Fed. Cir. 1997), Pitney Bowes, Inc. v. Hewlett-Packard Co., 182 F.3d 1298, 1305, 51 USPQ2d 1161, 1165 (Fed. Cir. 1999), and the like.)

-A scene graph data structure of a layered system for containing data that can be rendered into output that for subsequent integrated output that can be viewed; and (Itoh clearly teaches a scene graph data structure. Itoh teaches in 1:20-2:25 that clearly the system uses a scene graph; see Figure 8, element 101 ('scene graph'), where both the two dimensional and three dimensional data are fed into the scene graph representation data structure, note Figures 10-11, 13-14, etc. Further, Itoh teaches the use of scene graphs – see Figure 21 for an example of a scene graph (7:40-50) and the modification of their data using markup language (for example, the listed use VRML in 1:20-2:50,

22:19-26, etc, where XML and other languages are also taught) (19:1-20, 19:50-23,

22:33-55, finally 30:26-52, where it very clearly states that the scene graph provides an

API that can make many changes to the content, which matches with the above

system), and Itoh can receive data in SVG format as well (2:19-28). Next, clearly the

system of Itoh has many layers (see for example Figure 1 for a multi-layer scene

graph). Further, the system of Itoh has multiple layers (see Figures 2-5) in the sense

meant by applicant as discussed with respect to the MIL in the instant specification (see

the 'Definitions' section above, where Figure 2 of the instant specification shows a

structure similar to that shown in Figures 2-5 of Itoh. Finally, the resultant scene graph

is rendered into visible, viewable output in an integrated manner (indeed, Itoh clearly

combines three-dimensional and two-dimensional data into one scene graph, so that it

can be truly termed 'integrated'; see particularly Figure 8 (element 101) and also Figure

3. Finally, such output is rendered, see element 165 in Figure 8, where it is then shown

in web browser 3 (33:10-35))

--An object model including objects and other data that can be contained in the

scene graph data structure, at least some of the objects in the object model having

application program interfaces for invoking functions to modify contents of the scene

graph data structure. ((Itoh must prima facie modify data in the scene graph as it

creates various elements in it, can spatially translate them and perform various

operations on them as noted in the rejection to claim 1, and in 11:1-12:67, various

elements are shown as part of the markup include 11:55-60, where those transforms

are well known in the SVG art (see Steele as evidence that such transforms are well

known in the art and part of SVG)(SVG provides container objects for visual data, and

SVG clearly teaches the use of container objects, as in section 1.6 it clearly teaches the

use of 'container elements', which are defined as 'An element that can have graphic

elements and other container elements as child elements'. Clearly, the container object

could be the head object of the tree structure shown in Itoh Fig. 21, and the group node

referred to therein. Further SVG is known to modify elements in a hierarchy. Next,

clearly Figure 21 shows an object model – further, the model is discussed in Figure 9,

where there is a modeling coordinate system (element 92, labeled 'Scene Graph'), and

a modeling conversion unit, element 91. Therefore, clearly an object model is used,

and in Figure 21 the objects in such a model are illustrated. Some of the objects clearly

have APIs for invoking functions, particularly with respect to the elements and Java.

See for example Figure 12, where each object has various functions associated with it –

see for example where an animation for a shape can occur with various events, which

are known to have APIs. **See 22:50-55 – "Here, as described in detail later, a scene**

graph is data for a data structure of a non-cyclic graph composed of a tree-

shaped structure of nodes, and defined as an API in the above function.

Therefore, the scene graph of necessity must possess an API, as must the objects

within it, by virtue of inherited properties. Note next 30:33-55, where it is specified that

the scene graph provides an API capable of stipulating group nodes and leaf nodes,

and adding scene attributes which involve context, color of a shape or texture, and the

like)

Itoh uses SVG, and teaches the decomposition of graphic data into tree

structures in Fig. 20, with scene graph 160 being turned into a scene graph tree 158.

Further, Fig. 21 and 12 clearly illustrates what happens when the SVG animation

language is transformed into two sets of output language data. Clearly, as the SVG

standard sets forth, animation is done with SVG on a routine basis and the translation is

shown in Fig. 6. The tree of Figs. 12 and 21 is clearly a form of scene graph in the

broad definition set forth above and as discussed above. Thusly and *prima facie*

obviously, any 4:48-5:15, where animation would cause data in a scene graph to be

modified as the object was translated and the data structure containing locations and

other information would changed.

Reference Itoh thusly teaches all the limitations of the recited claim, except that it

does not expressly disclose that each object has its own API. However, since the

scene graph API obviously holds head and leaf nodes (), and the node objects are

contained in a tree-shaped data structure, where the data is stipulated as an API (33:4-

10), it would have been obvious to one of ordinary skill in the art at the time the

invention was made to modify the system such that each node had access to the

master API (and thusly had its own API) through inherited properties. Since the system

uses the Java API for final drawing purposes (33:12-35, 33:65-34:5), and Java has fully

inherited properties, e.g. each subclass has access to the APIs of the parent class and

the base Java classes, it would have been obvious to use the same structure with the

nodes in the scene graph. The motivation to make such a modification is found within

the knowledge of one of ordinary skill in the art, where such a one would be aware that

Figure 5, where the various types of media are integrated into the drawing commands

and the like, as required by the "high-level composition", which are **clearly** received via

the "two-dimensional drawing command" between elements 5 and 6 in Figure 5. The

"high-level means" would be the SVG commands (see 2:18-25 and the entire SVG

specification, which can clearly accommodate sound, video, and the like).

Finally, the recited "rendering means" comprise the three-dimensional canvas

105 in Figure 8 that sends graphics display data 165 to the web browser 3 to be

rendered using the Java AWT API and similar structures within the web browser, which

therefore constitutes 'rendering means' of the type set forth by the claim. Clearly, as

shown in Figures 8 and 9, particularly Figure 9, the scene graph is converted to world

space coordinates (steps 92/91), then translated to view space coordinates (steps 93-

95), then converted to view-space/viewport coordinates and normalized (steps 97/98),

where it is then sent as graphics data to be rendered (Figure 8).

Motivation/combination are incorporated by reference from claim 1 rejection.

Claims 5-17, 19, 21-27, 30-35, 37-64, and 66-67 are rejected under 35 U.S.C.

103(a) as unpatentable over Itoh in view of SVG as applied to claim 1, and further in

view of Steele.

As to claim 5,

The method of claim 4 further comprising, receiving a function call via an interface to

open the drawing visual for rendering, and in response, causing a device context to be

returned, the device context providing a mechanism for rendering the drawing visual.

making a language fully modular with inherited properties means that all code designed

in such a language will be reusable, and that the code would be modular, and thusly

much easier to implement (the reason for the design of Java as an improvement over

previous languages such as C and C++), as well as improving cross-platform portability

and making code execution platform-independent for the resulting graphics.

SVG is a markup language, therefore any SVG rendering utility would *prima facie*

receive markup, and any SVG rendering program would *prima facie* invoke such

functionality.

As to claim 65, this is a trivially obvious variant of claims 1 and 36.

The recited API means correspond to the API for assembling a scene graph and

the like (19:1-20, 19:50-23, 22:33-55, finally 30:26-52, where it very clearly states that

the scene graph provides an API that can make many changes to the content, which

matches with the above system, 33:5-10). Any software API is equivalent to another

software API. That API receives commands for drawing elements and modifying

elements within the scene graph as far as characteristics such as motion, color, texture,

the existence of various nodes, and the like. Further, SVG has its own APIs in addition

to the scene graph API mentioned above.

The recited "high-level composition means" comprises the SVG reader and such

(see Figure 20, where all the various three-dimensional and two-dimensional elements

feed into the scene graph; see Figure 14, where the VRML or XVL file is read in and

parsed; particularly the MMIDF function (see element 22 within element 5 in Figure 14),

which takes in commands and generates three-dimensional drawing commands; see

Reference Itoh does not expressly teach this limitation, but clearly teaches that users navigate through a virtual environment, which *prima facie* requires that transforms take place to visual objects and new visual objects be inserted (see rejections to claim 2 and 3). Reference Itoh does not expressly teach this limitation, but in 14:11-50 does teach that the system inserts two-dimensional drawing canvas commands into the three-dimensional drawing canvas by transforming them into three-dimensional commands.

Reference Steele teaches this limitation, as for example he teaches the insertion of unique identifiers into media streams [0106], and further [0088] that any visual element or object can modified. Such modifications and insertions *prima facie* must associate code with a suitable / desired insertion as that is the only way either a hierarchy of nodes (e.g. Fig. 7) or single nodes could be logically inserted.

For the second case, if the definition of context is the data associated with a specific element – e.g. the details of the element, its location, color, et cetera, these attributes are inherent in SVG elements as set forth in the rejections above, e.g. sections 11.1, 9.1-9.7, et cetera. Further, Steele teaches the same in Figure 7, where each element has certain properties that would be a drawing context, in the sense that each visual element has those properties associated with it [Steele 0052-0056 and 0059-0061].

As such, it would have been obvious to one having ordinary skill in the art at the time the invention was made to combine the SVG and graphics system of Steele and Itoh with the X3D and graphics system of Kim as set forth above, and because Itoh very

clearly teaches the use of SVG formatting and standards and further Steele would

extend the functionality of Itoh as Steele adds the behavior-based parsing capabilities

as shown in Figs. 3 and 5, for further control of elements – see [0052-0054]. (Further,

note that since this is performed by software, *prima facie* 'code' that is software

elements, would be invoked to perform any recited task.)

Reference Steele teaches this limitation, as for example he teaches the retrieval

of device context in [0101]. Clearly, the device receives information based on its device

context, which clearly is associated with the drawing context, as the two are one and the

same in this case. Steele teaches rendering in [0007 and 0011-0012]. The drawing

context per se is incorporated into the data structures of Steele (see Figure 7).

Reference Kim clearly teaches rendering in Fig. 2, element 143, "rendering means".

Further, Itoh teaches the use of scene graph data hierarchies (see Fig. 21), which

clearly establishes that drawing context must exist in order for that group node to know

its spatial location (29:60-30:25). It would have been obvious to one of ordinary skill in

the art at the time the invention was made to modify the system of Itoh to utilize a

device specific context so as to optimize data streamed to that device for purposes of

minimizing memory consumption (a large problem pointed out by Steele [0007]).

Further, note that since this is performed by software, *prima facie* 'code' that is software

elements (that is, specific functions), would be invoked to perform any recited task. In

any case, for rendering to occur, low-level graphics software **must** know details of the

underlying hardware in order to translate the high-level data to actual commands that

will cause the hardware to render the desired output.

As to claim 6,

The method of claim 1 further comprising, receiving brush data in association with the function call, and wherein causing data in the scene graph to be modified comprises invoking a brush function to modify a data structure in the scene graph data structure such that when a frame is rendered from the scene graph, an area will be filled with visible data corresponding to the brush data.

References Itoh, SVG, and Steele do teach this limitation by the use of SVG graphics. Turning to the SVG (, section 11 titled 'Painting: Filling, Stroking, and Marker Symbols', specifically section 11.1, 'With SVG, you can paint (e.g. stroke or fill) with: ..." and then proceeds to list several. The term 'brush data' is clearly analogous to the 'paint' operation in SVG with comparable data. Given that SVG allows (from section 11.1) a single color, a solid color (with or without opacity), a gradient, a pattern (vector or image), and custom patterns, clearly each visible element clearly has such data associated with it (see section 11.2 in its entirety, 11.7 for specific properties, section 11.8 for how painting properties can be inherited, which *prima facie* justifies the position that element have intrinsic painting properties, i.e. brush data as set forth above. Further, note that since this is performed by software, *prima facie* 'code' that is software elements, would be invoked to perform any recited task, and SVG data element *prima facie* and inherently possess paint data as set forth by the SVG specification above. As such, references Itoh, SVG, and Steele intrinsically teach this limitation and the SVG standard inherently handles these paint limitations. Motivation and combination is further taken from claim 5 and incorporated by reference. Clearly, these brush

commands would be received by the API and would cause changes in the scene graph as required, where the shape or area would be defined as a target of the brush operation via SVG, and the modified results would be rendered once the scene graph had been appropriately modified.

As to claim 7,

The method of claim 6 wherein the brush data comprises receiving data corresponding to a solid color.

References Itoh, SVG, and Steele do not expressly teach these limitations; references Itoh, SVG, and Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 11.1 of the SVG specification sets forth that a user can paint with a solid color with opacity, thus meeting this limitation. Further, note that since this is performed by software, *prima facie* 'code' that is software elements, would be invoked to perform any recited task, and SVG data element *prima facie* and inherently possess paint data as set forth by the SVG specification above. As such, references Itoh, SVG, and Steele intrinsically teach this limitation. Motivation and combination are further taken from claim 6 above and incorporated by reference.

As to claim 8,

The method of claim 6 wherein receiving brush data comprises receiving data corresponding to a linear gradient brush and a stop collection comprising at least one stop.

References Itoh, SVG, and Steele do not expressly teach these limitations; reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 11.1 of the SVG specification sets forth that a user can paint with a gradient that can be linear. Further, sections 11.7.1 and 11.7.2 of the specification sets forth that gradient stops are included in the SVG 'color-interpolation' property. Further, note that since this is performed by software, *prima facie* 'code' that is software elements, would be invoked to perform any recited task, and SVG data element *prima facie* and inherently possess paint data as set forth by the SVG specification above. As such, references Itoh, SVG, and Steele intrinsically teach this limitation; motivation and combination is taken from 6 and incorporated via reference.

As to claim 9,

The method of claim 6 wherein receiving brush data comprises receiving data corresponding a radial gradient brush.

References Itoh, SVG, and Steele do not expressly teach these limitations; references Itoh, SVG, and Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 11.1 of the SVG specification sets forth that a user can paint with a gradient that can be radial and also see sections 11.7.1 and 11.7.2 for more detail, thus meeting this limitation. Further, note that since this is performed by software, *prima facie* 'code' that is software elements, would be invoked to perform any recited task, and SVG data element *prima facie* and inherently possess paint data as set forth by the SVG specification above. As such, references

Itoh, SVG, and Steele intrinsically teach this limitation; motivation and combination is

incorporated from claim 6.

As to claim 10,

The method of claim 6 wherein receiving brush data comprises receiving data

corresponding to an image.

References Itoh, SVG, and Steele do not expressly teach these limitations;

references Itoh, SVG, and Steele teach them intrinsically as set forth above in claim 6

and reference SVG clearly supports this position. Section 11.1 of the SVG specification

sets forth that a user can paint with an image with further details provided in section

11.7.5 under the 'image-rendering' property. Further, note that since this is performed

by software, *prima facie* 'code' that is software elements, would be invoked to perform

any recited task, and SVG data element *prima facie* and inherently possess paint data

as set forth by the SVG specification above. Motivation and combination is incorporated

from claim 6.

As to claims 11 and 56,

The method of claim 10 further comprising, receiving markup corresponding to an

image effect to apply to the image.

References Itoh, SVG, and Steele do not expressly teach these limitations;

references Itoh, SVG, and Steele teaches them intrinsically as set forth above in claim 6

and reference SVG clearly supports this position. Section 14.4 of the SVG specification

sets forth that a user can use any image as an opacity mask, thus meeting this

limitation, given that alpha blending is *prima facie* an image effect. Further, note that

since this is performed by software, *prima facie* 'code' that is software elements, would

be invoked to perform any recited task, and SVG data element *prima facie* and

inherently possess paint data as set forth by the SVG specification above, and SVG is

inherently a markup language, so the rendering portion of Steele would receive such

information (the rendering functionality is inherent in SVG – see section 11.7, 14.4, et

cetera). Motivation and combination is also incorporated from claim 6.

As to claim 12,

The method of claim 1 further comprising, receiving pen data in association with the

function call, and wherein causing data in a scene graph structure to be modified

comprises invoking a pen function that defines an outline of a shape.

References Itoh, SVG, and Steele do not expressly teach these limitations;

references Itoh, SVG, and Steele teach them intrinsically as because of their use of

SVG as set forth above in the rejections to claims 4 and 5 and reference SVG clearly

supports this position. The term 'pen data' used by applicant above is comparable or

analogous to any set of data defining the outline of a shape, including SVG 'path' data.

Section 11.3 of the SVG specification sets forth that a user can fill a path that would

correspond to the outline of shape with multiple illustrations provided for this under the

'nonzero' and 'even odd' subheadings – see details on paths – with further details

provided in section 11.3 and 11.4 (the individual strokes that create these effects. SVG

is a markup language, therefore any SVG rendering utility would *prima facie* receive

markup. As such, references Itoh, SVG, and Steele intrinsically teach this limitation.

Motivation for combining Itoh and SVG is taken from the rejection to claim 1. Motivation for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 13,

The method of claim 1 wherein causing data in a scene graph data structure to be modified corresponds to invoking a geometry-related function to represent an ellipse in the scene graph data structure.

References Itoh, SVG, and Steele do not expressly teach these limitations; references Itoh, SVG, and Steele teach them intrinsically as set forth above in claim 1 and reference SVG clearly supports this position. The SVG specification sets forth classes of shapes in section 9.1, where all six of the recited shapes (rectangle, polygon, path, line, polyline, and ellipse) are set forth. Further, the SVG view in Steele decomposes SVG instructions into scene graphs containing basic SVG shapes as above [Steele 0052], where 'Visual Elements' include Shape classes. SVG is a markup language, therefore any SVG rendering utility would *prima facie* receive markup. As such, references Itoh, SVG, and Steele intrinsically teach this limitation. Motivation for combining Itoh and SVG is taken from the rejection to claim 1. Motivation for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 14,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a geometry-related function to represent a rectangle in the scene graph data structure.

References Itoh, SVG, and Steele do not expressly teach these limitations; references Itoh, SVG, and Steele teach them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. The SVG specification clearly shows in section 9.1 that rectangles are a basic shape, and further that in 9.2 under Example rect02 that such rectangles can have rounded corners, and code is provided that implements such. Also, the 'Rect' class inherently has geometry-related functions as set forth in the beginning to section 9.2. SVG is a markup language, therefore any SVG rendering utility would *prima facie* receive markup. As such, reference Steele shows a rectangle 715 in the scene graph in Figure 7 that intrinsically teaches this limitation. As such, references Itoh, SVG, and Steele intrinsically teach this limitation. Motivation for combining Itoh and SVG is taken from the rejection to claim 1. Motivation for adding Steele is taken from the rejections to claims 4 and 5. Also, said element can be animated under SVG section 19.2. Steele clearly teaches data modification in [0061] as set forth above.

As to claim 15,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a geometry-related function to represent a path in the scene graph data structure.

References Itoh, SVG, and Steele do not expressly teach these limitations; reference Steele teaches them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Further, Steele Fig. 6 shows an animation where path information is extracted into element 620, and listed in Fig. 7 [see Steele 0050 and

0079]. Further, the SVG specification sets forth path data in section 9.1 as existing and

how a 'path' can define a shape or similar. Both meanings are covered herein. Steele

clearly teaches data modification in [0061] as set forth above. Itoh teaches a path

element in 11:15-18.

SVG is a markup language, therefore any SVG rendering utility would *prima facie*

receive markup, and any SVG rendering program would *prima facie* invoke such

functionality. As such, references Itoh, SVG, and Steele intrinsically teach this

limitation. As such, references Itoh, SVG, and Steele intrinsically teach this limitation.

Motivation for combining Itoh and SVG is taken from the rejection to claim 1. Motivation

for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 16,

The method of claim 1 wherein causing data in the scene graph to be modified

comprises invoking a geometry-related function to represent a line in the scene graph

data structure.

References Itoh, SVG, and Steele do not expressly teach these limitations;

references Itoh, SVG, and Steele teach them intrinsically as set forth above in claim 6

and reference SVG clearly supports this position. Further, Steele Fig. 6 shows an

animation where path information is extracted into element 620, and listed in Fig. 7 [see

Steele 0050 and 0079]. A line element can be animated under SVG section 19.2, which

is obviously geometric. Line elements are set forth in SVG section 9.5, and their

geometric functions. Steele clearly teaches data modification in [0061] as set forth

above.

o   SVG is a markup language, therefore any SVG rendering utility would *prima facie* receive markup, and any SVG rendering program would *prima facie* invoke such functionality, and the scene graph shown in Figure 7 could clearly include lines since they are Visual Elements [Steele 0060-0061, which supports animation, et cetera]. As such, references Itoh, SVG, and Steele intrinsically teach this limitation. Motivation for combining Itoh and SVG is taken from the rejection to claim 1. Motivation for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 17,

The method of claim 1 wherein causing data in a scene graph data structure to be modified comprises invoking a function related to hit-testing a visual in the scene graph data structure.

References Itoh, SVG, and Steele do not expressly teach these limitations; references Steele and Itoh teach them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Clearly, Kim teaches three-dimensional navigation in Figure 4 – that is, navigation using a UI through a two-dimensional view-port of a three-dimensional environment, which is what any display normally shows. Therefore, given that a portable computer could clearly be used, the user would clearly be interacting with the display. As such, hit testing would be required for user interactivity, as could the system of Steele under the same rationale. The SVG specification sets forth hit testing in section 16.6 (the two paragraphs right at the end of the section) where hit testing (namely, hit detection) is taught, specifically testing text for character or cell hits and testing visual elements for hits in their entirety, and such

information is clearly communicated in markup language – see section 16.2 for event

types and elements transmitted in markup, for example. Steele clearly teaches data

modification in [0061] as set forth above.

SVG is a markup language, therefore any SVG rendering utility would *prima facie*

receive markup, and any SVG rendering program would *prima facie* invoke such

functionality. As such, references Itoh, SVG, and Steele intrinsically teach this

limitation. Motivation for combining Itoh and SVG is taken from the rejection to claim 1.

Motivation for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 19,

The method of claim 1 wherein causing data in a scene graph data structure to be

modified comprises invoking a function related to calculating a bounding box of a visual

in the scene graph data structure.

References Itoh, SVG, and Steele do not expressly teach these limitations;

references Itoh, SVG, and Steele teach them intrinsically as set forth above in claim 6

and reference SVG clearly supports this position. Bounding box calculations are taught

in section 7.1 and detailed in section 7.11 where they are calculated. Steele clearly

teaches data modification in [0061] as set forth above. In Steele, the scene graph

shown in Figure 7 could clearly include lines since they are Visual Elements [Steele

0060-0061, which supports animation, et cetera and would logically include bounding

boxes]. As such, references Itoh, SVG, and Steele intrinsically teach this limitation.

Motivation for combining Itoh and SVG is taken from the rejection to claim 1. Motivation

for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 21,

The method of claim 1 further comprising invoking a visual manager to render a tree of

at least one visual object to a rendering target.

References Itoh, SVG, and Steele do not expressly teach these limitations;

references Itoh, SVG, and Steele teach them intrinsically as set forth above in claim 6

and reference SVG clearly supports this position. Further, Steele Fig. 6 shows an

animation where path information is extracted into element 620, and listed in Fig. 7 [see

Steele 0050 and 0079] as trees. Further, Itoh teaches in Fig. 21 the use of trees as set

forth there, and teaches manager objects in 2:50-67. SVG teaches that all

implementations must implement a rendering model as set forth in 3.1 and so forth, and

scene graphs are known to directed acyclic, i.e. trees. Clearly, this model is

implemented through the DOM interfaces set forth in section 4, and each element has

its own element information that controls rendering aspects. Steele clearly teaches data

modification in [0061] as set forth above. It is *prima facie* obvious that a 'visual

manager' of some form must exist in order to handle formatting issues and precedence

in rendering, and Steele teaches such a manager in [0075-0076. Clearly the rendering

information each visual element [Steele 0056-0061] is sufficient such that it is its own

'rendering target' as set forth above. It would have been obvious to one of ordinary skill

in the art to add a manager if necessary to determine the precedence of item rendering

with respect to the tree. As such, references Itoh, SVG, and Steele intrinsically teach

this limitation. Motivation for combining Itoh and SVG is taken from the rejection to

claim 1. Motivation for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 22,

The method of claim 1 wherein causing data in the scene graph to be modified

comprises invoking a function to place a container object in the scene graph data

structure, the contained object configured to contain at least one visual object.

References Itoh, SVG, and Steele do not expressly teach these limitations;

references Itoh, SVG, and Steele teaches them intrinsically as set forth above in claim 6

and reference SVG clearly supports this position. Further, Steele Fig. 7 shows a tree

derived from an animation is shown – Figure 6 [see Steele 0050 and 0079]. SVG

clearly teaches the use of container objects, as in section 1.6 it clearly teaches the use

of 'container elements', which are defined as 'An element that can have graphic

elements and other container elements as child elements'. Steele clearly teaches data

modification in [0061] as set forth above. Clearly, the container object could be the

head object of the tree structure shown in Steele Fig. 7 or the group node of Itoh, shown

as the head node in Fig. 21. As such, references Itoh, SVG, and Steele intrinsically

teach this limitation. Motivation for combining Itoh and SVG is taken from the rejection

to claim 1. Motivation for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 23,

The method of claim 1 wherein causing data in the scene graph to be modified

comprises invoking a function to place image data into the scene graph data structure.

References Itoh, SVG, and Steele do not expressly teach these limitations;

references Itoh, SVG, and Steele teaches them intrinsically as set forth above in claim 6

and reference SVG clearly supports this position. Clearly, visual elements can be

covered by or tiled with images as established in SVG section 11.1, where SVG

teaches: "...can paint (i.e. fill or stroke) with: ...a pattern (vector or image, possibly

tiled) ..." which clearly establishes this, with more detail in section 11.7.5 and 11.12. As

such, references Itoh, SVG, and Steele intrinsically teach this limitation. Motivation for

combining Itoh and SVG is taken from the rejection to claim 1. Motivation for adding

Steele is taken from the rejections to claims 4 and 5.

As to claim 24,

The method of claim 23 wherein causing data in the scene graph to be modified

comprises invoking a function to place an image effect object into the scene graph data

structure that is associated with the image data.

References Itoh, SVG, and Steele do not expressly teach these limitations;

references Itoh, SVG, and Steele teaches them intrinsically as set forth above in claim 6

and reference SVG clearly supports this position. Section 14.4 of the SVG specification

sets forth that a user can use any image as an opacity mask for any visual element,

thus meeting this limitation, given that alpha blending is *prima facie* an image effect.

Steele clearly teaches data modification in [0061] as set forth above. As such,

references Itoh, SVG, and Steele intrinsically teach this limitation. Motivation for

combining Itoh and SVG is taken from the rejection to claim 1. Motivation for adding

Steele is taken from the rejections to claims 4 and 5.

As to claim 25,

The method of claim 1 wherein causing data in the scene graph to be modified

comprises invoking a function to place data corresponding to text into the scene graph

data structure.

References Itoh, SVG, and Steele do not expressly teach these limitations;

references Itoh, SVG, and Steele teach them intrinsically as set forth above in claim 6

and reference SVG clearly supports this position. Section 10.1 of the SVG specification

sets forth the use of a 'text' element, and Steele teaches the inclusion of text element

725 in the data tree shown in Fig. 7. Steele clearly teaches data modification in [0061]

as set forth above. Obviously, anything inserted into the tree of Itoh (Fig. 21) would

clearly place text in the scene. As such, references Itoh, SVG, and Steele intrinsically

teach this limitation. Motivation for combining Itoh and SVG is taken from the rejection

to claim 1. Motivation for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 26,

The method of claim 1 wherein causing data in the scene graph to be modified

comprises invoking a function to provide a drawing context in response to the function

call.

References Itoh and Steele do not expressly teach this limitation, but clearly

teaches that users navigate through a virtual environment, which *prima facie* requires

that device specific information regarding display information, e.g. drawing context, be

available to the rendering engine. Reference Steele teaches this limitation, as for

example he teaches the retrieval of device context in [0101]. Clearly, the device

receives information based on its device context, which clearly is associated with the

drawing context, as the two are one and the same in this case. For the second case, if the definition of context is the data associated with a specific element – e.g. the details of the element, its location, color, et cetera, these attributes are inherent in SVG elements as set forth in the rejections above, e.g. sections 11.1, 9.1-9.7, et cetera. Further, Steele teaches the same in Figure 7, where each element has certain properties that would be a drawing context, in the sense that each visual element has those properties associated with it [Steele 0052-0056 and 0059-0061]. Steele clearly teaches data modification in [0061] as set forth above. Itoh does not expressly teach this limitation.

SVG is a markup language, therefore any SVG rendering utility would *prima facie* receive markup, and any SVG rendering program would *prima facie* invoke such functionality. SVG is also a subset of XML, and SVG teaches metadata use in section 21.1. As such, references Itoh, SVG, and Steele intrinsically teach this limitation. Motivation for combining Itoh and SVG is taken from the rejection to claim 1. Motivation for adding Steele is taken from the rejections to claims 4 and 5.

It further would have been obvious to modify the system of Itoh to utilize a device specific context so as to optimize data streamed to that device for purposes of minimizing memory consumption (a large problem pointed out by Steele [0007]), and the SVG DOM interfaces in section 4.1-4.4 (SVG specification) clearly provide methods for retrieving drawing information, which would be that context.

As to claim 27,

The method of claim 26 wherein the function call corresponds to a retained visual, and further comprising, calling back to have the drawing context of the retained visual returned to the scene graph data structure.

References Itoh, SVG, and Steele do not expressly teach these limitations; references Steele and Itoh teach them intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Section 11.1 of the SVG specification clearly sets forth that all elements (as well as 3.1 and 4.2) have properties associated with them. The system of Steele clearly caches visuals during processing – see [0083], and it would be obvious that such data would be pulled from the cache to find out the state and properties of a visual element. Steele clearly teaches data modification in [0061] as set forth above. Further, it would be obvious to one of ordinary skill to so modify the Itoh reference to cache the visuals so that they would be retained and that data would be pulled from the cache as set forth above, and as the Steele reference sets forth to have it pulled from there during data processing, including that of data trees like unto the one in Figure 7, as in [0100 Steele]. As such, references Itoh, SVG, and Steele intrinsically teach this limitation. Motivation for combining Itoh and SVG is taken from the rejection to claim 1. Motivation for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 30,

The method of claim 1 wherein causing data in the scene graph to be modified comprises invoking a function to place animation data into the scene graph data structure.

References Itoh does not expressly teach this limitation, while reference Steele

does teach it. Steele in Figs. 6 and 9 shows animated elements [0041] and in Fig. 7

shows that each subgroup is shifted a certain amount with x and y coordinates given.

Steele [0050, 0052] for example provides that such animation takes place, and the SVG

standard in 19.1 – 19.5 clearly sets forth how each element can have animation

associated with it, which clearly is placed into the scene graph of Fig. 7. Therefore,

clearly animation data is put into the tree of Fig. 7 Steele, which is clearly a scene graph

by every known definition of the term, and a sample SVG XML program is provided in

the second page of Fig. 9. Motivation for combining Itoh and SVG is taken from the

rejection to claim 1. Motivation for adding Steele is taken from the rejections to claims 4

and 5.

As to claim 31,

The method of claim 30 further comprising communicating timeline information

corresponding to the animation data to another layer of the media integration layer.

References Steele and Itoh do not expressly teach this limitation – reference Kim

does teach rendering (143) and scene processing (144) means in Fig. 2, which clearly

can perform compositing (since the system does 3D rendering), while reference Steele

does teach the animation limitation. Steele clearly establishes in [0051-0054] and Figs.

6 and 9 that animation takes place through the SVG standard. Section 19.2 of SVG

sets forth how this takes place, and at the bottom three paragraphs of section 19.2.2 it

clearly states that animation has a document start and document end, and further in the

second to last paragraph that the SVG system indicates the timeline position of

document fragments being animated. Further, according to SVG 19.2.2 the animation

is by document fragment and object path, which clearly are passed to the system is

specified in, for example, the second page of Fig. 9 in the SVG XML program. Clearly,

the system of Steele performs compositing and rendering [0007, 0011-0012], as does

Kim. Finally, reference SVG teaches that it supports compositing (section 14.2.1), and

the X3D specification supports compositing (6.2.3 for example). The composition

engine would be, for example, the web browser 3 and Java AWT API of Itoh as in

Figure 8 and the like. Clearly the compositional engine (and/or OpenGL) constitutes

another layer of the rendering engine (the 'low-level portion' contemplated by applicant's

specification). Motivation for combining Itoh and SVG is taken from the rejection to

claim 1. Motivation for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 32,

The method of claim 31 wherein the composition engine interpolates graphics data

based on the timeline to animate an output corresponding to an object in the scene

graph data structure.

References Itoh and SVG do not expressly teach this limitation, while reference

Steele does teach it. Steele in Figs. 6 and 9 shows animated elements [0041] and in

Fig. 8, clearly during an animation the actions are shown, where the system in steps

835, 840, and 845 performs interpolation for nodes shown in the tree in Fig. 7. Clearly

interpolation takes place during animation [0072, 0077-0079] which performs

interpolation during the animation process as set forth in the SVG standard, and *prima*

facie the output would only be objects in the scene graph, and they would *prima facie* be based on the timeline as set forth in the rejection to claim 31 above.

SVG is a markup language, therefore any SVG rendering utility would *prima facie* receive markup, and any SVG rendering program would *prima facie* invoke such functionality. As such, reference Steele intrinsically teaches this limitation and it would have been obvious to one having ordinary skill in the art at the time the invention was made to combine the X3D and graphics system of Steele with the SVG and graphics system of Kim as set forth above and the SVG standard specification. (Please see also rejection to claim 1 for additional motivation).

As to claim 33,

The method of claim 1 wherein receiving a function call via an interface of a media integration layer comprises receiving markup, and wherein causing data in a scene graph data structure to be modified comprises parsing the markup into a call to an interface of an object.

Clearly, the system of Itoh receives markup in the form of XML, VRML, and SVG (2:1-65, 4:1-65) commands, and clearly these are two- and three-dimensional drawing commands (Figure 8) at a high level that are acted upon by the system to add objects to the scene graph via the scene graph API (30:1-65), where this clearly constitutes receiving markup (SVG and VRML are clearly XML, which is a markup language).])(Steele inherently, because SVG graphics data requires DOM objects (see SVG standard), which inherently requires an XML parser as part of that system, as the XML data has to be processed and formatted, which is what a parser *by definition* does – as

shown in Fig. 3, where the SVG comes into element 210 and then enters the SVG DOM

305 for translation from the SVG reader to the SVG compiler, which *prima facie* must

have a parser (all compilers must parse syntax at a minimum – this is a fundamental of

the computer art)).

As to claims 34 and 55,

The method of claim 1 wherein causing data in the scene graph to be modified

comprises invoking a function to place an object corresponding to audio and/or video

data into the scene graph data structure.

References Itoh, SVG, and Steele do teach it. Itoh teaches this limitation in Fig.

5. Steele in Figs. 8 shows audio elements 820 and 830 in the animation execution and

in Fig. 7 a scene graph data structure (a tree). Steele [0050, 0052] for example

provides that such animation takes place, and the SVG standard in 6.18 clearly sets

forth aural style sheets, that are audio data that each element can have animation

associated with it, which clearly is placed into the scene graph of Fig. 7. Also, by

definition, SVG animations would be video. Motivation for combining Itoh and SVG is

taken from the rejection to claim 1. Motivation for adding Steele is taken from the

rejections to claims 4 and 5.

As to claim 35,

The method of claim 1 wherein causing data in the scene graph to be modified

comprises changing a mutable value of an object in the scene graph data structure.

References Kim does not expressly teach this limitation, while reference Steele

does teach it. Steele teaches in [0014] that one embodiment of his invention changes

the visual graph in accordance to changes of the sequence graph, where the visual graph is comparable to the "scene graph" of applicant and mutable values (e.g. position) of elements in the tree are shifted as per Steele [0052-0057]. Therefore, the limitation is met, and it would have been obvious to modify it so that it in fact change mutable values on the tree if applicant feels that this is not an adequate embodiment of this particular limitation. Motivation for combining Itoh and SVG is taken from the rejection to claim 1. Motivation for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 37,

The system of claim 36 wherein at least one function is invoked to place a tree of visual objects into the scene graph data structure.

References Itoh and SVG do not expressly teach this limitation, while reference Steele does teach it. Further, Steele Fig. 6 shows an animation where path information is extracted into element 620, and listed in Fig. 7 [see Steele 0050 and 0079] as trees. SVG teaches that all implementations must implement a rendering model as set forth in 3.1 and so forth, and scene graphs are known to directed acyclic, i.e. trees. Clearly, Steele [0064] teaches that the visual graph is changed according to the addition or alteration of elements in the sequence graph, which clearly would happen with the insertion of new visible objects into the tree shown in Fig. 7, i.e. during animation when new elements were being shown. It would have been obvious to modify the system of Steele to so perform the recited task. Motivation for combining Itoh and SVG is taken from the rejection to claim 36. Motivation for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 38,

The system of claim 37 further comprising a visual manager that when invoked renders the tree of visual objects to a rendering target.

Reference Steele teaches these limitations intrinsically as set forth above in claim 6 and reference SVG clearly supports this position. Further, Steele Fig. 6 shows an animation where path information is extracted into element 620, and listed in Fig. 7 [see Steele 0050 and 0079] as trees. SVG teaches that all implementations must implement a rendering model as set forth in 3.1 and so forth, and scene graphs are known to directed acyclic, i.e. trees. Clearly, this model is implemented through the DOM interfaces set forth in section 4, and each element has its own element information that controls rendering aspects. Steele clearly teaches data modification in [0061] as set forth above. It is *prima facie* obvious that a 'visual manager' of some form must exist in order to handle formatting issues and precedence in rendering, and Steele teaches such a manager in [0075-0076. Clearly the rendering information each visual element [Steele 0056-0061] is sufficient such that it is its own 'rendering target' as set forth above.

Specifically, the arrangement of a tree in the scene graph data structure again would be dependent on the insertion of items into the visual tree, and obviously given that Steele and SVG teach container objects as taught in the rejection to claim 36 above, the insertion of a container object would *prima facie* cause the insertion of a tree, and arranging a tree in the scene graph would qualify as "altering" elements within the system of Steele and falls within [0064].

It would have been obvious to modify the systems of Kim and Itoh in light of Steele to so perform the recited task. SVG is a markup language, therefore any SVG rendering utility would *prima facie* receive markup, and any SVG rendering program would *prima facie* invoke such functionality and perform the recited conversion. Motivation for combining Itoh and SVG is taken from the rejection to claim 36. Motivation for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 39,

The system of claim 37 wherein the markup is converted to a method call to place a visual collection object into the scene graph data structure.

References Itoh and SVG do not expressly teach this limitation, while reference Steele does teach it. Reference Itoh teaches it implicitly since Itoh generates tree objects (2:50-67, Figs. 12 and 21). Further, Steele Fig. 6 shows an animation where information is extracted and separated into specific elements, and shown / listed in Figs. 7 and 8 [see Steele 0050 and 0079] as trees. SVG teaches that all implementations must implement a rendering model as set forth in 3.1 and so forth, and scene graphs are known to directed acyclic, i.e. trees. Clearly, Steele [0064] teaches that the visual graph is changed according to the addition or alteration of elements in the sequence graph, which clearly would happen with the insertion of new visible objects into the tree shown in Fig. 7, i.e. during animation when new elements were being shown. A collection object would clearly encompass a container object containing a tree of visual objects; examiner is interpreting the definition in this way.

Specifically, the arrangement of a tree in the scene graph data structure again

would be dependent on the insertion of items into the visual tree, and obviously given

that Steele and SVG teach container objects as taught in the rejection to claim 36

above, the insertion of a container object would *prima facie* cause the insertion of a tree,

and arranging a tree in the scene graph would qualify as "altering" elements within the

system of Steele and falls within [0064].

It would have been obvious to modify the system of Itoh in light of Steele and

SVG to so perform the recited task. Motivation for combining Itoh and SVG is taken

from the rejection to claim 36. Motivation for adding Steele is taken from the rejections

to claims 4 and 5.

As to claim 40,

The system of claim 36 wherein at least one function is invoked to place a visual object

into the scene graph data structure.

Reference Itoh does not expressly teach this limitation, but clearly teaches that

users navigate through a virtual environment, which *prima facie* requires that transforms

take place to visual objects and new visual objects be inserted (see rejections to claim 2

and 3). Reference Steele teaches this limitation, as for example he teaches the

insertion of unique identifiers into media streams [0106], and further [0088] that any

visual element or object can modified. Such modifications and insertions *prima facie*

must associate code with a suitable / desired insertion as that is the only way either a

hierarchy of nodes (e.g. Fig. 7) or single nodes could be logically inserted.

For the second case, if the definition of context is the data associated with a

specific element – e.g. the details of the element, its location, color, et cetera, these

attributes are inherent in SVG elements as set forth in the rejections above, e.g.

sections 11.1, 9.1-9.7, et cetera. Further, Steele teaches the same in Figure 7, where

each element has certain properties that would be a drawing context, in the sense that

each visual element has those properties associated with it [Steele 0052-0056 and

0059-0061]. Motivation for combining Itoh and SVG is taken from the rejection to claim

36. Motivation for adding Steele is taken from the rejections to claims 4 and 5. (Further,

note that since this is performed by software, *prima facie* 'code' that is software

elements, would be invoked to perform any recited task.)

As to claim 41,

The system of claim 40 wherein the markup is converted to a method call to associate a

brush with the visual object.

This claim is a substantial duplicate of claim 6. The only difference is that the

markup is converted to a method call to perform the recited action, whereas claim 6

merely 'invokes a function' to perform the recited task, and as such would a trivial

variation of ordinary skill in the art. The rejection to claim 6 is herein incorporated by

reference with the motivation and combination. The SVG standard is referenced there

so there is no difference in the reference hierarchy and it would have been trivially

obvious to combine as set forth because the SVG specification is extensively cited by

and key to the Steele reference. Motivation for combining Itoh and SVG is taken from

the rejection to claim 36. Motivation for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 42,

The system of claim 40 wherein the markup is converted to a method call to associate a geometry with the visual object.

References Itoh, SVG, and Steele do not expressly teach these limitations; references Steele and Itoh teach them intrinsically as set forth above in the rejection to claim 36 and reference SVG clearly supports this position. The SVG specification sets forth classes of shapes in section 9.1, where all six of the recited shapes (rectangle, polygon, path, line, polyline, and ellipse) are set forth. Further, the SVG view in Steele decomposes SVG instructions into scene graphs containing basic SVG shapes as above [Steele 0052], where 'Visual Elements' include Shape classes. SVG is a markup language, therefore any SVG rendering utility would *prima facie* receive markup. Secondarily, the association of a geometry with a visual object would be intrinsic to the existence of such an object. However, SVG in section 11.4 sets forth different shapes of strokes and the same in 11.6 for markers. Therefore, it would have been obvious to one of ordinary skill in the art to modify the Kim reference in light of Steele to be able to change the type of geometry associated with a visual element in accordance with the teachings of SVG 11.4 and 11.6, which meets the above-recited limitations.

As such, references Steele and Itoh intrinsically teaches this limitation and it would have been obvious to one having ordinary skill in the art at the time the invention was made to combine the graphics system of Steele with the SVG standard

specification. (Please see also rejection to claim 36 for additional motivation).

Motivation for combining Itoh and SVG is taken from the rejection to claim 36.

Motivation for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 43,

The system of claim 42 wherein the geometry comprises at least one of a set containing

an ellipse geometry, a rectangle geometry, a line geometry and a path geometry.

The SVG specification sets forth classes of shapes in section 9.1, where all six of

the recited shapes (rectangle, polygon, path, line, polyline, and ellipse) are set forth.

Further, the SVG view in Steele decomposes SVG instructions into scene graphs

containing basic SVG shapes as above [Steele 0052], where 'Visual Elements' include

Shape classes.

This claim is a substantial duplicate of claim 13. Further, the term 'geometry'

used here is analogous to the 'shape' reference in claim 13. The only difference is that

the markup is converted to a method call to perform the recited action, whereas claim

13 merely 'invokes code' to perform the recited task, and as such would a trivial

variation of ordinary skill in the art. As such, the rejection for claim 13 is hereby

incorporated by reference with motivation and combination and is valid without further

comment. The six components of the SVG standard shapes include an ellipse, a

rectangle, a line, and a path – see SVG section 9.1. Motivation for combining Itoh and

SVG is taken from the rejection to claim 36. Motivation for adding Steele is taken from

the rejections to claims 4 and 5.

As to claim 44,

The system of claim 40 wherein the markup is converted to a method call to associate a transform with the visual object.

This claim is a substantial duplicate of claim 3. Further, the term 'geometry' used here is analogous to the 'shape' reference in claim 3. The only difference is that the markup is converted to a method call to perform the recited action, whereas claim 3 merely 'invokes code' to perform the recited task, and as such would a trivial variation of ordinary skill in the art. As such, the rejection for claim 3 is hereby incorporated by reference with motivation and combination and is valid without further comment. The six components of the SVG standard shapes include an ellipse, a rectangle, a line, and a path – see SVG section 9.1. Motivation for combining Itoh and SVG is taken from the rejection to claim 36. Motivation for adding Steele is taken from the rejections to claims 4 and 5. Further, the teachings of Steele are applicable because SVG is used herein to illustrate this point.

As to claim 45,

The system of claim 44 wherein the transform comprises a rotate transform for changing a perceived angle of the visual object.

This claim is a substantial duplicate of claim 3. Further, the term 'geometry' used here is analogous to the 'shape' reference in claim 3. The only difference is that the markup is converted to a method call to perform the recited action, whereas claim 3 merely 'invokes code' to perform the recited task, and as such would a trivial variation of ordinary skill in the art. Specifically, Steele [0053] clearly states that rotations are applied to visual objects or groups of visual objects, and states that as well.

As such, the rejection for claim 3 is hereby incorporated by reference with

motivation and combination and is valid without further comment. The six components

of the SVG standard shapes include an ellipse, a rectangle, a line, and a path – see

SVG section 9.1. Motivation for combining Itoh and SVG is taken from the rejection to

claim 36. Motivation for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 46,

The system of claim 44 wherein the transform comprises a scale transform for changing

a perceived size of the visual object.

Reference Itoh does not expressly teach this limitation, but clearly teaches that

users navigate through a virtual environment, which *prima facie* requires that transforms

take place to visual objects. References Steele and Itoh teach this limitation implicitly,

as he discloses rotations in [0053] and further states that rotations and other

transformations can be applied to an entire tree of objects, e.g. Fig. 7, and further [0088]

that any visual element or object can modified. Such modifications *prima facie* must

associate code with a suitable / desired transform (e.g. scaling, rotation, et cetera

[0053]), as that is the only way either a hierarchy of nodes (e.g. Fig. 7) or single nodes

could be scaled. Reference Itoh discloses rotations and other actions on SVG and XML

elements in 11:55-57.

Specifically, however, the SVG specification provides for scaling transformations

in Example Rotate-Scale in section 7.4 under coordinate system transformations

As such, It would have been obvious to one having ordinary skill in the art at the

time the invention was made to combine the SVG and graphics system of Steele and

the SVG specification, and because they provide for coordinate transforms as set forth

above, which inherently requires graphics transforms and such. (Please see claim 36

for additional motivation / combination justification, which is herein incorporated by

reference). (Further, note that since this is performed by software, *prima facie* 'code'

that is software elements, would be invoked to perform any recited task.)

As to claim 47,

The system of claim 44 wherein the transform comprises a translate transform for

changing a perceived position of the visual object.

This claim is a substantial duplicate of claim 3. Further, the term 'geometry' used

here is analogous to the 'shape' reference in claim 3. The only difference is that the

markup is converted to a method call to perform the recited action, whereas claim 3

merely 'invokes code' to perform the recited task, and as such would a trivial variation of

ordinary skill in the art. Specifically, Steele [0053] clearly states that translations are

applied to visual objects or groups of visual objects, and states that as well.

As such, the rejection for claim 3 is hereby incorporated by reference with

motivation and combination and is valid without further comment. The six components

of the SVG standard shapes include an ellipse, a rectangle, a line, and a path – see

SVG section 9.1. Motivation for combining Itoh and SVG is taken from the rejection to

claim 36. Motivation for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 48,

The system of claim 44 wherein the transform comprises a skew transform for changing

a perceived skew of the visual object.

Reference Itoh does not expressly teach this limitation, but clearly teaches that users navigate through a virtual environment, which *prima facie* requires that transforms take place to visual objects. References Steele, Itoh, and SVG teach this implicitly, that is Steele teaches it implicitly, this limitation that is, as he discloses rotations in [0053] and further states that rotations and other transformations can be applied to an entire tree of objects, e.g. Fig. 7, and further [0088] that any visual element or object can modified. Such modifications *prima facie* must associate code with a suitable / desired transform (e.g. scaling, rotation, et cetera [0053]), as that is the only way either a hierarchy of nodes (e.g. Fig. 7) or single nodes could be scaled.

Specifically, however, the SVG specification provides for skew transformations in Example Skew in section 7.4 under coordinate system transformations.

As such, It would have been obvious to one having ordinary skill in the art at the time the invention was made to combine the X3D and graphics system of Steele with the SVG and graphics system of Kim as set forth above and the SVG specification, and because they provide for coordinate transforms as set forth above, and the Kim reference inherently performs transforms as a user can navigate through a virtual 3D environment, which inherently requires graphics transforms and such. (Please see claim 36 for additional motivation / combination justification, which is herein incorporated by reference). (Further, note that since this is performed by software, *prima facie* 'code' that is software elements, would be invoked to perform any recited task.) Motivation for combining Itoh and SVG is taken from the rejection to claim 36. Motivation for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 49,

The system of claim 44 further comprising animation information associated with the

transform, and wherein the animation information causes transformation data

associated with the transform to change over time thereby animating the transformation

of the visual object over time.

Reference Itoh does not expressly teach this limitation, while references Steele,

Itoh, and SVG do teach it. Steele clearly establishes in [0051-0054] and Figs. 6 and 9

that animation takes place through the SVG standard. Section 19.2 of SVG sets forth

how this takes place, and at the bottom three paragraphs of section 19.2.2 it clearly

states that animation has a document start and document end, and further in the

second to last paragraph that the SVG system indicates the timeline position of

document fragments being animated. Further, according to SVG 19.2.2 the animation

is by document fragment and object path, which clearly are passed to the system is

specified in, for example, the second page of Fig. 9 in the SVG XML program. Clearly,

the system of Steele performs compositing and rendering [0007, 0011-0012], as does

Kim. Finally, reference SVG teaches that it supports compositing (section 14.2.1), and

the X3D specification supports compositing (6.2.3 for example). The composition

engine would be, for example, elements 143, 144, and 147 of Fig. 2 of Kim.

Steele in Figs. 6 and 9 shows animated elements [0041] and in Fig. 8, clearly

during an animation the actions are shown, where the system in steps 835, 840, and

845 performs interpolation for nodes shown in the tree in Fig. 7. Clearly interpolation

takes place during animation [0072, 0077-0079] which performs interpolation during the

animation process as set forth in the SVG standard, and *prima facie* the output would only be objects in the scene graph, and they would *prima facie* be based on the timeline as set forth above. SVG is a markup language, therefore any SVG rendering utility would *prima facie* receive markup, and any SVG rendering program would *prima facie* invoke such functionality. Motivation for combining Itoh and SVG is taken from the rejection to claim 36. Motivation for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 50,

The system of claim 40 wherein at least one function is invoked to associate a color with the visual object.

This claim is a substantial duplicate of claim 7. The only difference is that the markup is converted to a method call to perform the recited action, whereas claim 3 merely 'invokes code' to perform the recited task, and as such would a trivial variation of ordinary skill in the art. A color as recited here is applied via the paint method (SVG 11.1 – 11.4) and is the exact same as the "solid color" associated with a "brush" as in claim 7 – the "brush" is merely a semantic description as set forth above. As such, the rejection for claim 7 is hereby incorporated by reference with motivation and combination and is valid without further comment. Clearly, SVG teaches the use of a solid color with a visual object – see SVG 11.1. Motivation for combining Itoh and SVG is taken from the rejection to claim 36. Motivation for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 51,

The system of claim 40 wherein at least one function is invoked to associate gradient data with the visual object.

This claim is a substantial duplicate of claim 7. The only difference is that the markup is converted to a method call to perform the recited action, whereas claim 3 merely 'invokes code' to perform the recited task, and as such would a trivial variation of ordinary skill in the art. A gradient as recited here is applied via the paint method (SVG 11.1 – 11.4) and is the exact same as the "solid color" associated with a "brush" as in claim 7 – the "brush" is merely a semantic description as set forth above, and a linear gradient is certainly a gradient. As such, the rejection for claim 7 is hereby incorporated by reference with motivation and combination and is valid without further comment. Clearly, SVG teaches the use of a gradient with a visual object – see SVG 11.1. Motivation for combining Itoh and SVG is taken from the rejection to claim 36. Motivation for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 52,

The system of claim 40 wherein at least one function is invoked to associate a tile brush with the visual object.

Reference Steele does teach this limitation by the use of SVG graphics. Turning to the SVG (, section 11 titled 'Painting: Filling, Stroking, and Marker Symbols', specifically section 11.1, 'With SVG, you can paint (e.g. stroke or fill) with: ..." and then proceeds to list several. The term 'brush data' is clearly analogous to the 'paint' operation in SVG with comparable data. Given that SVG allows (from section 11.1), among other things a pattern (vector or image) that can be tiled, clearly each visible

element clearly has such data associated with it (see section 11.2 in its entirety, 11.7 for

specific properties, section 11.8 for how painting properties can be inherited, which

prima facie justifies the position that element have intrinsic painting properties, i.e. brush

data as set forth above. Further, note that since this is performed by software, *prima*

facie 'code' that is software elements, would be invoked to perform any recited task, and

SVG data element *prima facie* and inherently possess paint data as set forth by the

SVG specification above. As such, reference Steele intrinsically teaches this limitation.

Motivation for combining Itoh and SVG is taken from the rejection to claim 36.

Motivation for adding Steele is taken from the rejections to claims 4 and 5 and the SVG

standard inherently handles these paint limitations.

As to claim 53,

The system of claim 40 wherein at least one function is invoked to associate an image

with the visual object.

This claim is a substantial duplicate of claim 10. The only difference is that the

markup is converted to a method call to perform the recited action, whereas claim 3

merely 'invokes code' to perform the recited task, and as such would a trivial variation of

ordinary skill in the art. An image as recited here is applied via the paint method (SVG

11.1 – 11.4) and is the exact same as the image associated with a "brush" as in claim 7

– the "brush" is merely a semantic description as set forth above. As such, the rejection

for claim 10 is hereby incorporated by reference with motivation and combination and is

valid without further comment. Clearly, SVG teaches the use of an image with a visual

object – see SVG 11.1. Motivation for combining Itoh and SVG is taken from the

rejection to claim 36. Motivation for adding Steele is taken from the rejections to claims

4 and 5.

As to claim 54,

The system of claim 40 wherein at least one function is invoked to associate an image

with the visual object.

References Itoh, SVG, and Steele do not expressly teach these limitations;

references Itoh, SVG, and Steele teach them intrinsically as set forth above in claim 6

and reference SVG clearly supports this position. Section 11.1 of the SVG specification

sets forth that a user can paint with an image with further details provided in section

11.7.5 under the 'image-rendering' property. Further, note that since this is performed

by software, *prima facie* 'code' that is software elements, would be invoked to perform

any recited task, and SVG data element *prima facie* and inherently possess paint data

as set forth by the SVG specification above. Motivation and combination is incorporated

from claim 6.

As to claim 55,

The system of claim 40 wherein at least one function is invoked to associate a drawing

comprising drawing primitives with the visual object.

References Itoh, SVG, and Steele do not expressly teach these limitations;

references Steele and Itoh teach them intrinsically as set forth above in claim 40 and

reference SVG clearly supports this position. The SVG specification sets forth classes

of shapes in section 9.1, where six shapes are set forth, and they are fundamental

drawing primitives. Also, in SVG 7.11, the table lists a "primitiveUnits" that stand for

numeric primitives within elements of the basic classes or elements being referenced. .

Further, the SVG view in Steele decomposes SVG instructions into scene graphs

containing basic SVG shapes as above [Steele 0052], where 'Visual Elements' include

Shape classes. SVG is a markup language, therefore any SVG rendering utility would

prima facie receive markup. As such, reference Steele intrinsically teaches this

limitation. Motivation for combining Itoh and SVG is taken from the rejection to claim 36.

Motivation for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 56,

The system of claim 40 wherein at least one function is invoked to associate audio

and/or video media with the visual object.

References Itoh and SVG do not expressly teach this limitation, while reference

Steele does teach it. Steele in Figs. 8 shows audio elements 820 and 830 in the

animation execution and in Fig. 7 a scene graph data structure (a tree). Steele [0050,

0052] for example provides that such animation takes place, and the SVG standard in

6.18 clearly sets forth aural style sheets, that are audio data that each element can

have animation associated with it, which clearly is placed into the scene graph of Fig. 7.

Also, by definition, SVG animations would be video. SVG is a markup language,

therefore any SVG rendering utility would *prima facie* receive markup, and any SVG

rendering program would *prima facie* invoke such functionality. Motivation for

combining Itoh and SVG is taken from the rejection to claim 36. Motivation for adding

Steele is taken from the rejections to claims 4 and 5.

As to claim 57,

The system of claim 40 at least one function is invoked to associate an image effect with the visual object.

This claim is a substantial duplicate of claim 11. The only difference is that the markup is converted to a method call to perform the recited action, whereas claim 3 merely 'invokes code' to perform the recited task, and as such would a trivial variation of ordinary skill in the art. An image effect as recited here is applied as set forth in SVG section 14.4, with the alpha blending of course being an image effect. As such, the rejection for claim 11 is hereby incorporated by reference with motivation and combination and is valid without further comment, as the references are the same. Clearly, SVG teaches the use of an image with a visual object – see SVG 14.4. Also, the rejection to the parent claim is herein incorporated by reference.

As to claim 58,

The system of claim 40 wherein at least one function is invoked to associate a pen with the visual object, to describe how a shape is outlined.

This claim is a substantial duplicate of claim 12. The only difference is that the markup is converted to a method call to perform the recited action, whereas claim 12 merely 'invokes code' to perform the recited task, and as such would a trivial variation of ordinary skill in the art. . As such, the rejection for claim 12 is hereby incorporated by reference with motivation and combination and is valid without further comment, as the references are the same. Clearly, SVG teaches the use of a pen to set outlines – see SVG 11.4 and 11.5. Motivation for combining Itoh and SVG is taken from the rejection to claim 36. Motivation for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 59,

The system of claim 40 wherein at least one function is invoked to obtain a drawing context associated with the visual object.

This claim is a substantial duplicate of claim 26. The only difference is that the markup is converted to a method to call to perform the recited action, whereas claim 12 merely makes a 'function call' to perform the recited task, and as such would a trivial variation of ordinary skill in the art. As such, the rejection for claim 26 is hereby incorporated by reference with motivation and combination and is valid without further comment, as the references are the same. Motivation for combining Itoh and SVG is taken from the rejection to claim 36. Motivation for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 60,

The system of claim 40 wherein at least one function is invoked to associate hit testing data with the visual object.

This claim is a substantial duplicate of claim 17. The only difference is that the markup is converted to a method to call to perform the recited action, whereas claim 17 merely 'causes data to be modified' to perform the recited task, and as such would a trivial variation of ordinary skill in the art. As such, the rejection for claim 17 is hereby incorporated by reference with motivation and combination and is valid without further comment, as the references are the same. Motivation for combining Itoh and SVG is taken from the rejection to claim 36. Motivation for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 61,

The system of claim 40 wherein at least one function is invoked to associate a rectangle with the visual object.

This claim is a substantial duplicate of claim 14. The only difference is that the markup is converted to a method to call to perform the recited action, whereas claim 14 merely 'causes data to be modified' to perform the recited task, and as such would a trivial variation of ordinary skill in the art. As such, the rejection for claim 14 is hereby incorporated by reference with motivation and combination and is valid without further comment, as the references are the same. Also, clearly whether or not the visual object is in the scene graph does not matter in the context of this specific claim as it is in claim 14. Motivation for combining Itoh and SVG is taken from the rejection to claim 36. Motivation for adding Steele is taken from the rejections to claims 4 and 5.

As to claim 62,

The system of claim 61 wherein at least one function is invoked to describe how a source rectangle should be stretched to fit a destination rectangle.

Reference Itoh does not expressly teach this limitation, but clearly teaches that users navigate through a virtual environment, which *prima facie* requires that transforms take place to visual objects. Reference Steele teaches this implicitly limitation, as he discloses rotations in [0053] and further states that rotations and other transformations can be applied to an entire tree of objects, e.g. Fig. 7, and further [0088] that any visual element or object can modified. Such modifications *prima facie* must associate code

with a suitable / desired transform (e.g. scaling, rotation, et cetera [0053]), as that is the

only way either a hierarchy of nodes (e.g. Fig. 7) or single nodes could be scaled.

Specifically, however, the SVG specification provides for scaling transformations

in Example Rotate-Scale in section 7.4 under coordinate system transformations.

Further, SVG clearly teaches the use of rectangles as basic shapes and provides for

methods for associating one shape with a visual object as set forth in SVG 9.1.

Obviously from the parent claim, a rectangle could be scaled using the methods set

forth in the SVG standard listed immediately above and it would have been obvious to

one ordinary skill in the art to so modify the apparatus of Kim in light of Steele and SVG

as set forth above.

Motivation for combining Itoh and SVG is taken from the rejection to claim 36.

Motivation for adding Steele is taken from the rejections to claims 4 and 5. (Please see

claim 61 for additional motivation / combination justification, which is herein

incorporated by reference).

As to claims 63 and 64,

The system of claim 61 wherein at least one function is invoked to describe how content

is positioned {horizontally | vertically} within a container corresponding to the visual

object.

The system of Steele clearly teaches the objects know their location, both before

and during animation and placement (see Figures 7, 8, and all pages of Figure 9, where

x and y coordinates are listed and known by the various constituent objects.

As to claim 66,

The system of claim 65 wherein the rendering means includes low-level composition

means for constructing a frame for viewing based on data received from the high-level

composition engine.

Steele in Fig. 15 shows that various programs, including the operating system,

are on the flash memory, which in [0136] is specified to contain all the low-level

programs of the operating system – graphics is low-level functionality. Since there is no

specific graphics unit, all graphics operations and compositing are done by the

operating system in the microprocessor, which *prima facie* means that in that

embodiment, such graphics are done at a low-level, that is the rendering is done by the

operating system at a low level. The scene graph is high-level in that it is embodied in

RAM and is held as an abstraction – this is a function of the SVG standard that keeps

tree nodes and container nodes as abstract as possible, therefore the embodiment in

Fig. 18 must do the same. In any case, low-level composition means that it would be

done by hardware that is much faster than software. As such, it would be obvious to

modify the device of Kim and Steele to use low-level rendering, and in any case Steele

has the rendering means set forth in the rejection to claim 32 above, which is *prima*

facie entirely hardware.

As to claim 67,

The system of claim 65 further comprising animation means, the high-level composition

engine providing timeline data to the low-level composition means for interpolating the

appearance of visible data across at least two frames to animate the visible data over

time.

Steele does teach this limitation. Steele in Figs. 6 and 9 shows animated

elements [0041] and in Fig. 8, clearly during an animation the actions are shown, where

the system in steps 835, 840, and 845 performs interpolation for nodes shown in the

tree in Fig. 7. Clearly interpolation takes place during animation [0072, 0077-0079]

which performs interpolation during the animation process as set forth in the SVG

standard, and *prima facie* the output would only be objects in the scene graph, and they

would *prima facie* be based on the timeline as set forth in the rejection to claim 31

above. Clearly this constitutes 'interpolation' as required above, because the high-level

engine does not perform this part of the interpolation. Note Itoh provides interpolator

nodes for objects in the scene graph (24:65-25:25) and passes the data on to the lower

level compositional unit, in terms of timeline data, as set forth in the previous rejection to

claim 31, the rejection to which is incorporated by reference in its entirety.

Conclusion

Any inquiry concerning this communication or earlier communications from the

examiner should be directed to Eric Woods whose telephone number is 571-272-7775.

The examiner can normally be reached on M-F 7:30-5:00.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's

supervisor, Ulka Chauhan can be reached on 571-272-7782. The fax phone number for

the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

Eric Woods   5/26/06

ULKA CHAUHAN
SUPERVISORY PATENT EXAMINER